

AD--A257 311



NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

X WINDOW APPLICATION EXTENSION
WITH
THE ANDREW TOOLKIT
by

Jeffrey J. Stenzoski
September, 1992

Thesis Advisor:

Balasubramaniam Ramesh

Approved for public release; distribution is unlimited

92-29910



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE					
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b. OFFICE SYMBOL (If applicable) 37		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000			7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			Program Element No.	Project No.	Task No.
					Work Unit Accession Number
11. TITLE (Include Security Classification) X WINDOW APPLICATION EXTENSION WITH THE ANDREW TOOLKIT (UNCLASSIFIED)					
12. PERSONAL AUTHOR(S)					
13a. TYPE OF REPORT Master's Thesis		13b. TIME COVERED From To		14. DATE OF REPORT (year, month, day) September, 1992	
				15. PAGE COUNT 55	
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17. COSATI CODES			18. SUBJECT TERMS (continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUBGROUP	X Windows, Andrew Toolkit, REMAP, ConceptBase, GraphBrowser		
19. ABSTRACT (continue on reverse if necessary and identify by block number) This thesis investigates the extension of an X11 Windows-based application using the high-level Andrew Toolkit to permit direct knowledge base access via a graphical user interface (GUI). Programming with Andrew Toolkit is relatively straightforward, once initial familiarity with the toolkit structure and methodology is achieved.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS REPORT <input type="checkbox"/> DTIC USERS				21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL B. Ramesh				22b. TELEPHONE (Include Area code) (408) 646-2439	
				22c. OFFICE SYMBOL AS/RA	

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsoleteSECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

X Window Application Extension
with
The Andrew Toolkit

by

Jeffrey J. Stenzoski
Lieutenant Commander, United States Navy
B.S., United States Naval Academy

Submitted in partial fulfillment
of the requirements for the degree of

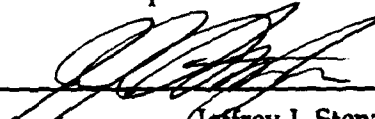
MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

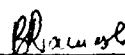
NAVAL POSTGRADUATE SCHOOL

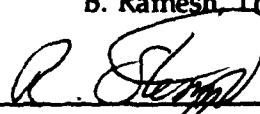
September 1992


Author:


Jeffrey J. Stenzoski

Approved by:


B. Ramesh, Thesis Advisor


Roger Stemp, Thesis Co-advisor


David R. Whipple, Jr., Chairman
Department of Administrative Sciences

ABSTRACT

This thesis investigates the extension of an X11 Windows-based application using the high-level Andrew Toolkit to permit direct knowledge base access via a graphical user interface (GUI). Programming with Andrew Toolkit is relatively straightforward, once initial familiarity with the toolkit structure and methodology is achieved.

DTIC QUALITY INSPECTED 4

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	GENERAL	1
B.	BACKGROUND	2
C.	THESIS OBJECTIVES	3
D.	SCOPE	3
E.	ORGANIZATION OF THE STUDY	3
II.	X WINDOW BASICS	5
A.	GENERAL	5
B.	ARCHITECTURE	5
	1. The Window Manager	7
C.	TOOLKITS	8
III.	THE ANDREW TOOLKIT	10
A.	INTRODUCTION	10
B.	THE OBJECT-ORIENTED ENVIRONMENT	11
C.	DYNAMIC LOADING	13
	1. <i>runapp</i>	13
	2. Benefits of Dynamic Loading	13
D.	BASIC TOOLKIT OBJECTS: DATA OBJECT AND VIEW	14
	1. Inset Data Storage	14
E.	EVENT PROCESSING	15

1. The Interaction Manager	15
2. The View Tree	15
IV. THESIS PROJECT ENVIRONMENT	18
A. THESIS OBJECTIVE	18
B. THE REMAP PROJECT	18
1. REMAP Model Prototype Environment	20
a. GraphBrowser	21
V. EXTENDING THE GRAPHBROWSER INSET	23
A. <i>GraphBrowser</i> Inset Components	23
B. WRITING NEW ANDREW CLASS FILES	24
1. Class Header File	24
2. Class C File	25
C. MODIFICATION OF <i>.graphbrowserinit</i> FILE	26
1. Menu Item Description	27
2. Graphical Type Description	30
D. MODIFICATION OF REMAP MODEL	32
E. COMPILATION OF THE ANDREW CLASS	35
1. Setting Environment Variables	36
2. Creating the <i>Imakefile</i>	36
3. Installation of Project Files	37
4. <i>Makefile</i> Generation and Code Compilation	38
VI. CONCLUSIONS AND RECOMMENDATIONS	39
A. HIGH-LEVEL VERSUS LOW-LEVEL TOOLKITS	39

B. SELECTION OF THE ANDREW TOOLKIT	39
C. LEARNING ANDREW	40
1. Prerequisite Skills	40
2. Andrew-specific Skills	41
D. CLOSING REMARKS	42
APPENDIX	43
LIST OF REFERENCES	46
BIBLIOGRAPHY	47
INITIAL DISTRIBUTION LIST	48

I. INTRODUCTION

A. GENERAL

The advent of bit-mapped graphics workstations, from the Macintosh to the Sun, has changed the face of interactive computing. The overwhelming user acceptance of windowed graphics and point-and-click command entry has spurred the development of hardware-specific graphical user interface (GUI) application software. The need to develop a different graphical interface for each hardware vendor was overcome by the X Window system (Jones, 1989, p. 1). The virtue of X is its portability across hardware, software, and network boundaries.

Graphical applications which intermix various methods of data representation--bit mapped images, text, video, animation, etc. - are typically costly to build, hard to debug, or slow to run, depending on the tools used to build them (Borenstein, 1990, p. 1). Andrew Toolkit (ATK) is a C-based, object-oriented, user-interface toolkit designed specifically to support the development of stand alone multimedia applications. Running under the X Windows environment, ATK allows the development of exciting multi-media applications which can be seamlessly ported across hardware, software, and networks.

This thesis research is based on programming experience gained while converting an application from the Sun View window environment to X Windows using Andrew Toolkit.

B. BACKGROUND

The problem and expense of developing a different graphical interface for each hardware vendor's product was tackled by two large users of workstations from multiple vendors: Project Athena and the Laboratory for Computer Science, both at the Massachusetts Institute of Technology (Jones, 1989, pp. 1-2). This research led to the development of the X Window system. Early versions of X were designed and implemented primarily by Robert Scheifler and Ron Newman of MIT, as well as Jim Gettys from the Digital Equipment Corporation (Young, 1990, p. 1). In January 1987, a dozen computer manufacturers agreed to support the standardization of the X Window system by forming the X Consortium. Today, the X Consortium consists of hundreds of members and provides a forum to facilitate the development of X extensions that meet various emergent needs. The current MIT X Windows distribution, X11R5 (release 5), contains the Andrew Toolkit, developed by Carnegie Mellon University and the IBM Corporation at the Information Technology Center (Borenstein, 1990, p. xiii).

C. THESIS OBJECTIVES

This thesis examines the extension of an X-based application, *GraphBrowser*, using the object-oriented, high-level Andrew Toolkit. This extension will permit the direct editing of objects from the REMAP (**RE**presentation and **MA**intenance of **P**rocess knowledge) model using a graphical user interface (GUI). Specific techniques for extending Andrew applications in the X usage environment are discussed.

D. SCOPE

The scope of this thesis is limited to a brief overview of the X Window system's structure and a more detailed description of a high-level toolkit application extension. Low-level toolkits, such as Xlib, are neither discussed nor demonstrated.

E. ORGANIZATION OF THE STUDY

Beyond this introduction and the later conclusions, this thesis consists of four major chapters. Chapter II contains a brief introduction to the X Window system, providing a basic understanding of the underlying windowing environment. Chapter III examines peculiar aspects of the Andrew Toolkit, including the object-oriented environment. Chapter IV discusses the various elements of the REMAP project that form the context of this thesis. The REMAP project is introduced along with the *ConceptBase* knowledge base management system

(in which the REMAP prototype system is implemented) and the *GraphBrowser* application. Finally, Chapter V focuses on the specific steps necessary to develop and integrate the Andrew extension into the X usage environment.

II. X WINDOW BASICS

A. GENERAL

The emergence of X Window as the de facto user interface standard is traceable to a number of vendor-relevant factors:

1. The industry is eager for standards
2. X is distributed by a neutral source (MIT)
3. The source code is free
4. X is restricted to defining windowing mechanisms, not policies for interface styles (Upton 1990).

The X Window interface enables the workstation world to transition from sole command-line prompt entry to the WIMP (windows, icons, menus, and pointers) model while creating a particular look and feel for their application. One important difference between X and other windowing systems is that X does not define nor enforce any one interface style, but instead simply provides the mechanisms to support a variety of user-designed interface styles (Young, 1990, p. 2). In other words, X concentrates on the skeleton and leaves the clothing to the customer (Upton 1990).

B. ARCHITECTURE

The architecture of the X Window system is based on the client-server model. Unfortunately the definition of "client" and "server" in the X world are exactly the reverse of the

terminology used in the mini computer and LAN environments (Upton 1990).

REMOTE X CLIENTS

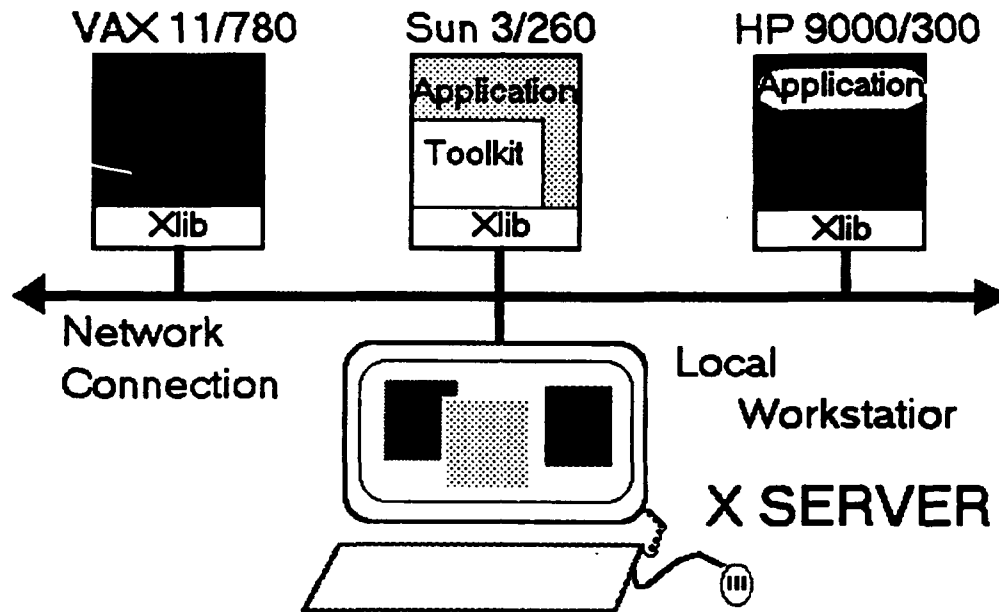


Figure 1. The X client-server model (Young 1990).

In X jargon, the server is a single process running typically on a workstation or personal computer with a graphics display. The server provides a portable layer between all applications and the display hardware and is responsible for creating and manipulating windows, producing text and graphics, and handling input events from the keyboard and mouse. An application that utilizes the display and input handling capabilities of the X server is known as a client. The client and server communicate via a network connection with one of many network protocols, such as TCP/IP, DECnet and

Chaos. Any client can communicate with any server, provided they both obey the X protocol (Young, 1990, p. 2). Figure 1 depicts the X Window client-server relationship.

1. The Window Manager

The window manager is a special client application which controls the placement, sizing and appearance of the windows on the server terminal. Window managers typically ask the server to redirect requests involving the structure of an X window to the window manager rather than letting the server act on the request directly (Young, 1990, p. 10). Additionally, window managers may provide a distinct "look and feel" by decorating windows with three-dimensional frames complete with titles, scroll bars and push buttons (Jones, 1989, p. 10).

Another important responsibility of the window manager is to act as an intermediary between clients which coexist on the same screen (Barkakati, 1991, p. 32). Coordination between the clients and between the clients and the server is facilitated by the protocol defined in the Inter-client Communication Conventions Manual (ICCCM), prepared by David Rosenthal of Sun Microsystems (Barkakati, 1991, p. 36).

The most popular commercial window managers are OSF/Motif and OPEN LOOK. The OSF/Motif window manager, *mwm*, was derived from work done by Hewlett-Packard and Microsoft, and has a look and feel similar to Microsoft Windows

(Barkakati, 1991, p. 34). The X11 distribution comes with several window managers, including *uwm*, based on popup menus, or *wm*, a window manager that decorates windows with banners and buttons (Jones, 1989, p. 10). X11 also comes with the Tabbed Window Manager, *twm*, formerly known as "Tom's Window Manager."

C. TOOLKITS

The X application generally communicates with the X network protocol through one or more levels of toolkits. X toolkits are pre-packaged libraries of C language subroutines designed to aid the developer by hiding the details of implementing graphical objects such as buttons, slide bars and menus. Toolkits exist at three distinct levels. The most widely used low-level interface to X is the standard C language library known as Xlib. Xlib defines a set of functions that provide access and control over the display, windows and input devices (Young, 1990, p. 11). Built on top of Xlib is a middle level toolkit known as Xt Intrinsics, or simply "Xt." Also known as the "X Toolkit," Xt was developed primarily by Digital Equipment Corporation and MIT's project Athena (Jones, 1989, pp. 2-3). Xt Intrinsics is designed to support a set of user interface components known as widgets. Widget sets such as Motif and Andrew are considered high-level toolkits and provide a rich collection of GUI design objects from which to develop an application. Figure 2 depicts the

general structure of an X application. As mentioned earlier, the X protocol assures device independence and provides an interface between client and server.

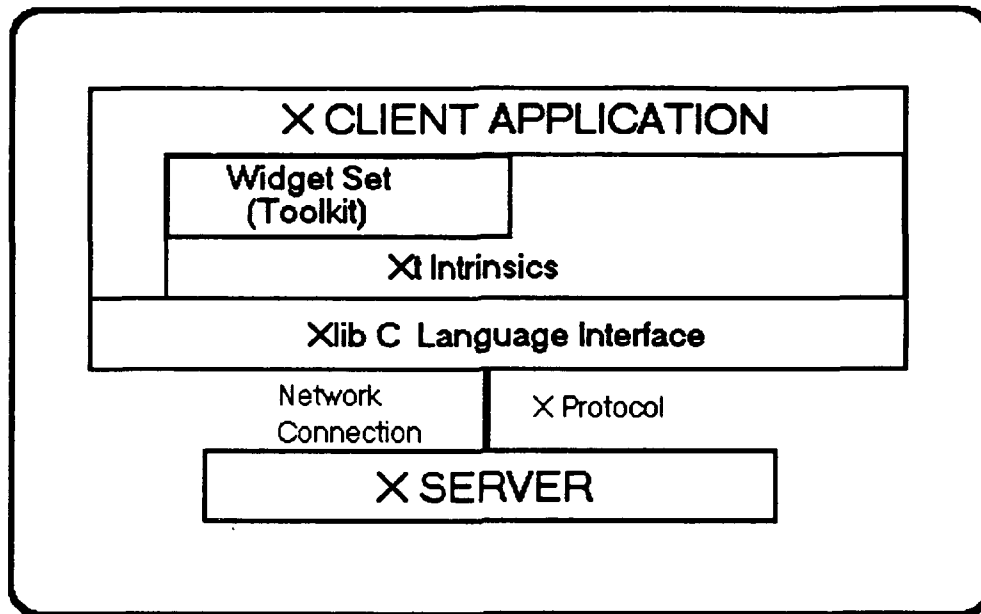


Figure 2. Structure of an X application (adapted from Young 1990).

III. THE ANDREW TOOLKIT

A. INTRODUCTION

In 1982, an organization known as the Information Technology Center at Carnegie Mellon University set out to develop what is known as the Andrew System, named after the university's two major benefactors, Andrew Carnegie and Andrew Mellon. The Andrew System consists of three main components: 1) The Andrew Message System, 2) The Andrew Help System, and 3) The Andrew Toolkit and Application Programs. This chapter will focus on The Andrew Toolkit specifically.

The Andrew Toolkit (ATK) is a user interface toolkit with two main goals: (1) to support the development of stand-alone applications that integrate text, graphics, and all images in a standard, efficient user interface, and (2) to support the development of multi-media editors. The ATK was built using an object-oriented system called the Andrew Class System, and was designed to provide a foundation on which a large number of diverse user-interface applications can be developed (Palay, 1988, p. 1). The Toolkit is written in the C language, using a preprocessor to provide an object-oriented environment and dynamic linking of code. The major thrust of the ATK design has been to simplify the creation of multimedia applications which allow entirely different types of

independently generated media to be intermingled fluidly in a single application (Borenstein, 1990, p. 2).

B. THE OBJECT-ORIENTED ENVIRONMENT

The ATK is built using the Andrew Class System (Class). Class was modeled after the C++ object-oriented environment and permits the definition of object methods and class procedures (Palay, 1988, p. 7). Class is a C language-based system consisting of a small run-time library and preprocessor. An Andrew Class is composed of two files: The standard C file (".c") containing the class data and methods, and a class header file (".ch") containing the class specification (see Chapter V, section B for an example class header file). The Andrew Class preprocessor generates two files from the class header file, an exported header file (".eh") used when defining a class, and an imported header file (".ih") used by any other code that wants to use the class (Borenstein, 1990, p. 18). The .c source files are not run through the preprocessor, and look similar to ordinary C files. Next, from the ".c", ".eh", and ".ih" files, the standard C-language compiler (cc) generates the usual object file (".o"), which is processed by a program called makedo. The output of makedo is the dynamic object file (".do"), which is dynamically loaded on request at run-time. The entire compilation process is shown in Figure 3 (Borenstein, 1990, pp. 20-21).

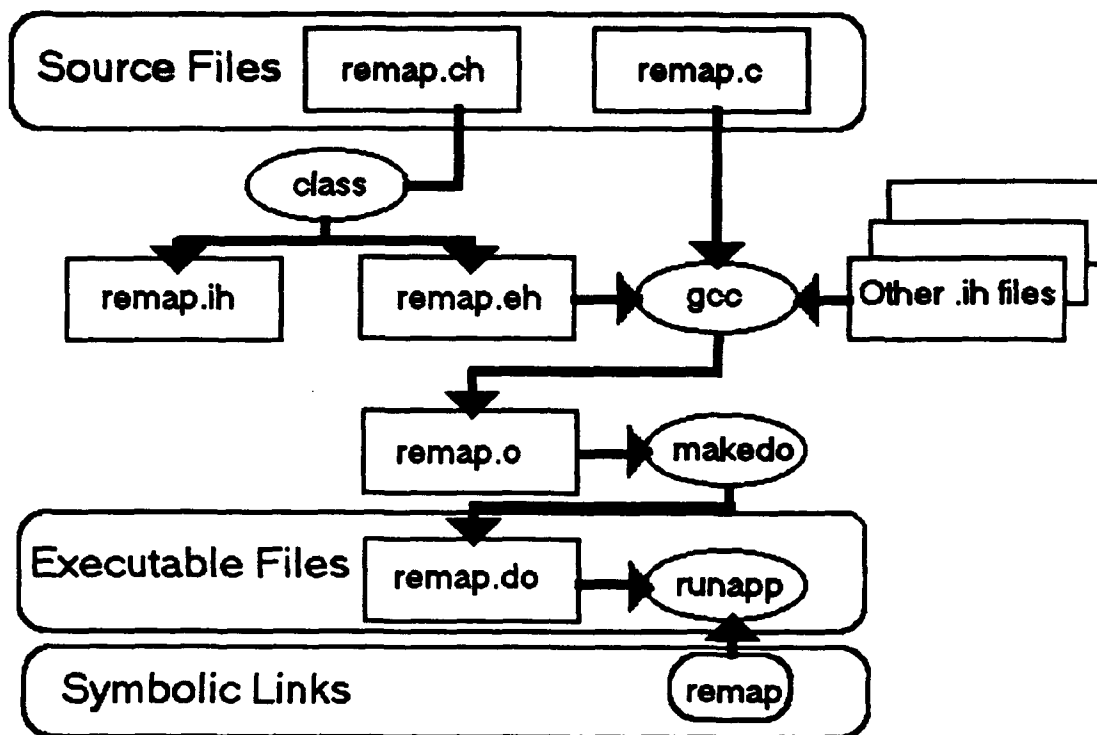


Figure 3. ATK compilation process (from Borenstein 1990).

The object-oriented nature of Andrew is generated by this peculiar preprocessing and compilation procedure. The object-oriented nature of ATK enables developers to benefit from class inheritance when creating specialized objects from the rich set of objects provided with the ATK distribution, as well as from the application development environment. For instance, REMAP uses objects developed by the *ConceptBase* system on which it is based. The following paragraphs describe the main elements of the Andrew object-oriented environment.

C. DYNAMIC LOADING

In conventional C programs, a linking loader must be run to create a single binary executable program. In contrast, Andrew's dynamic loading feature allows the loading of arbitrary objects that users request after the program has started executing.

1. *runapp*

The main workhorse of ATK is a single binary program called "*runapp*," consisting of all the main ATK objects. *Runapp* can be thought of as an "upside-down library"; that is, instead of dynamically loading the toolkit into the Andrew application, the *runapp* binary is run first, then the application dynamically loaded into it (Borenstein, 1990, pp. 23-24). When *runapp* executes, it first determines the name it was called by; if other than "*runapp*" it will add the suffix "*app*" to the application name ("*remap-app*," for example), and try to dynamically load a class called "*remap-app*" found in the dynamic object file, "*remap.do*." Finally, the compilation process forms a symbolic link between the application name and the Toolkit itself (see Figure 3).

2. Benefits of Dynamic Loading

The dynamic loading feature of ATK has three main benefits. First, application development is streamlined by the elimination of the linking process. Second, it promotes code sharing and reuse, since new objects are readily

available to all developers without the need to recompile them. Finally, it enhances the extensibility of ATK as a whole by allowing significant modification to the system without the need to recompile the Toolkit itself (Borenstein, 1990, p. 21).

D. BASIC TOOLKIT OBJECTS: DATA OBJECT AND VIEW

Two of the most important objects provided by the Andrew Toolkit are data objects and views. A data object contains the actual information to be displayed, while the view contains the details of how the data is to be displayed and how the user will be able to manipulate and interact with the data. The basic ATK component is made up of a data object/view pair known as an inset. In a window containing a raster image, for example, the raster data object will contain the lines that make up the image, shading, etc., whereas the raster view will provide the exact methods for drawing the image on the screen and for handling various input events such as keyboard entry and mouse click.

1. Inset Data Storage

The storage of data associated with the data object and view is handled differently. Unlike the data object, the information associated with a view is considered useful only while the application is running, and cannot be stored in a file between sessions. Views do, however, provide print capability within the Andrew Toolkit (Palay, 1988, p. 3).

The distinction between data object and view allows multiple, simultaneous representations of the same data object. Additionally, editing performed in one view will be reflected in all views, since they all share the same data object. This separation also provides a highly modularized structure that facilitates application development (Palay, 1988, p. 3).

E. EVENT PROCESSING

1. The Interaction Manager

The ATK is an event-driven system. The coordination of event handling for a window is performed by a core Toolkit object called the interaction manager, or *im*. When an application creates a window, it does so by generating an interaction manager to be the top-level object in the window (Borenstein, 1990, pp. 36-37). The *im* translates input events such as key strokes, mouse, menu, and exposure events from the underlying window system to the view objects contained within that window. In addition, the *im* is responsible for synchronizing drawing requests between views and for hiding the input model used by the underlying window system, namely X11 or the Andrew window manager, *wm*.

2. The View Tree

Views within a window are organized in a tree structure, with the interaction manager at the root. The view tree is the tree of view objects as they are configured in a

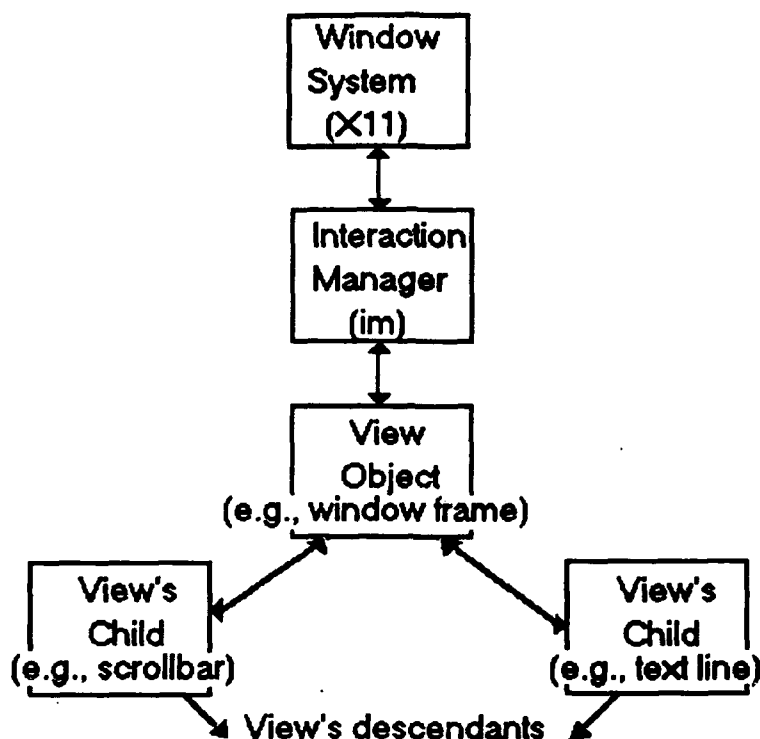


Figure 4. Window view tree structure.

window at run time (Borenstein, 1990, p. 40). The *im* has one child view, which, in turn, can have any number of children. When an event is received by the *im* by the underlying window system, the *im* passes the event down to its child view which then determines if it should handle the event or pass it down to one of its children. This process recurs until some view in the view tree finally handles the event (Palay, 1988, p.4). This relationship, depicted in Figure 4, is a distinctive characteristic of ATK. Other toolkits rely on the physical relationship of components on a screen to determine event handling, sometimes resulting in the blockage of event transmission to hidden or partially obscured components.

Furthermore, some toolkits use a global analysis of all views in order to process and distribute events, whereas ATK delegates this authority to each view over its children (Palay, 1988, p. 6).

IV. THESIS PROJECT ENVIRONMENT

A. THESIS OBJECTIVE

The basic objective of this thesis project was to research the implementation of a C-language extension to the ATK-based *GraphBrowser* utility program to enable the retrieval and manipulation of objects from the REMAP model in the X Window environment.

B. THE REMAP PROJECT

The focus of the REMAP project (**RE**presentation and **MA**intenance of **P**rocess knowledge) is the structured capture of design rationale, or "process knowledge," during the requirements engineering phase of a software development project. The REMAP conceptual model includes the Issue Based Information Systems method (IBIS). IBIS was used at Microelectronic Computer technology Corporation (MCC) in the Design Journal research project as a way of representing design deliberations in large design projects (Ramesh 1992). The IBIS method utilizes a set of three primitives and relationships among them in a rhetorical model for representing the "argumentation" process (Ramesh 1992). This initial primitive set was expanded in REMAP based on an empirical study of experienced systems analysts using a requirements engineering exercise (see Table 1).

TABLE 1. EXPANSION OF IBIS PRIMITIVES FOR REMAP MODEL

Initial IBIS Model Primitives	Additional REMAP Model Components
Issue Position Argument	Requirement Constraint Design Object Assumption Decision

Figure 5 represents the basic relationship of the elements in the conceptual REMAP model.

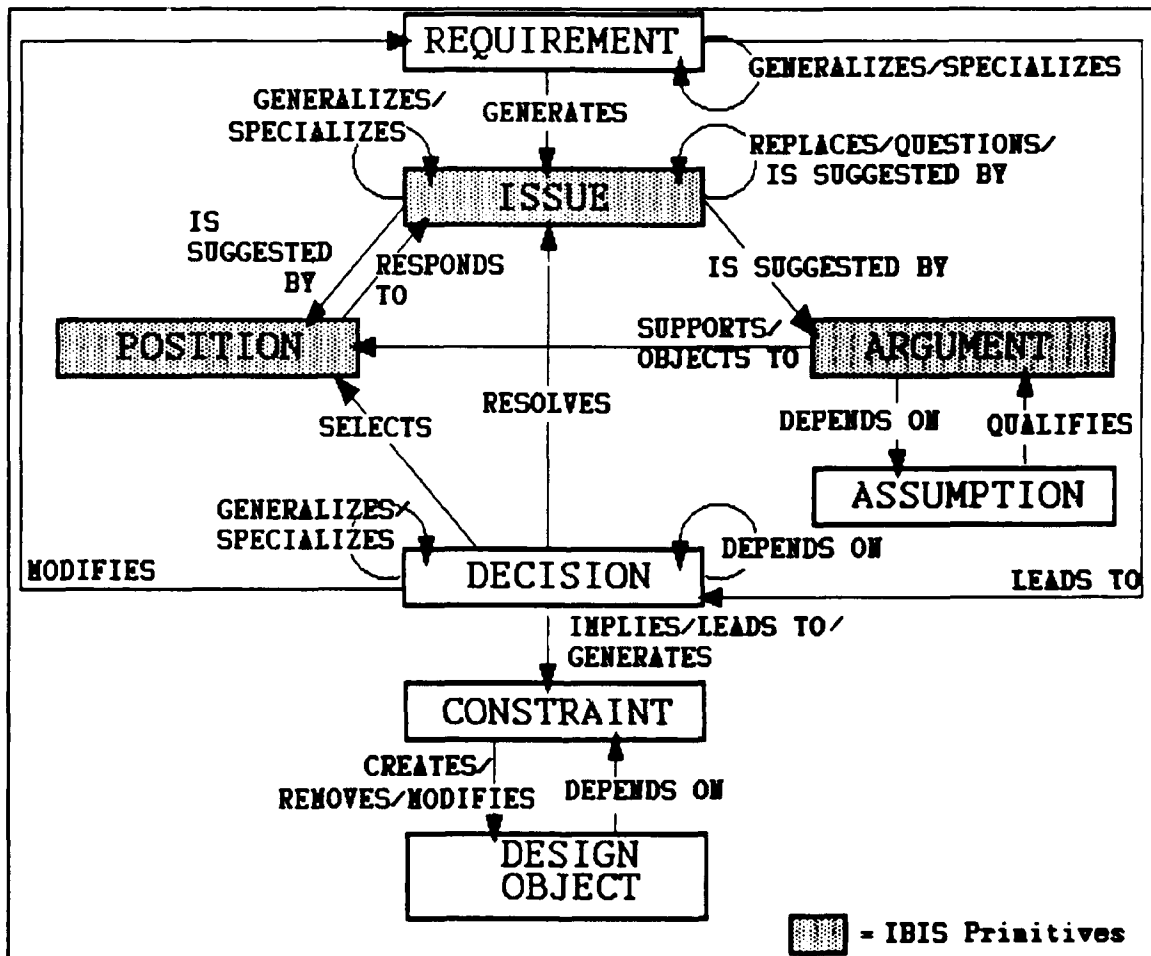


Figure 5. The conceptual REMAP model (Ramesh 1992).

1. REMAP Model Prototype Environment

The REMAP prototype is implemented in *ConceptBase*--an experimental knowledge base management system developed at the University of Passau (Jarke, 1991, p. 1). *ConceptBase* manages the REMAP knowledge base which is expressed in the knowledge representation language Telos. Telos is a high-level, conceptual modeling language which "integrates a thoroughly axiomatized structurally object-oriented kernel with a predicative assertion language in the style of deductive databases and with a temporal sublanguage that covers validity as well as transaction time" (Jarke, 1991, p. 2). *ConceptBase* is designed as a coordination mechanism for heterogeneous design environments and can run distributed over local or wide area networks with the Internet protocol in a client-server architecture (Jarke, 1991, p. i). It is important to note that the client-server definitions for *ConceptBase* are the reverse of those used in the X Windows context described in Chapter II. As a standard client, *ConceptBase* supports both Sun View and X Windows (X11) usage environments. The X11 usage environment was written in C utilizing the Andrew Toolkit and includes a number of tools such as the Telos editor and the *GraphBrowser* application described below (Jarke, 1991, p. 3).

a. *GraphBrowser*

The *GraphBrowser* application is a window-based utility which allows a point-and-click method of graphically browsing the contents of a model loaded from the *ConceptBase* server and displays the contents in the form of a directed acyclic graph (DAG). The top-level *GraphBrowser* menu options shown in Table 2 allow the expansion and removal of Telos-defined objects displayed within the *GraphBrowser* window.

TABLE 2. TOP-LEVEL GRAPHBROWSER MENU FUNCTIONS

GraphBrowser Menu Option	Function
- erase node	Removes selected object from display (does not modify contents of knowledge base).
- any	Expands graph with selected object.
- show attributes	Displays all direct attributes of selected objects.
- show instances	Displays all direct instances of selected objects.
- show subclasses	Displays one level of subclasses of selected objects.

GraphBrowser functionality is limited, however, since it does not provide for actual modifications to the server's knowledge base, such as the insertion, deletion, or editing of an object instance. Furthermore, *GraphBrowser*-generated server queries regarding object instances and attributes are formatted according to the predefined Telos

knowledge base classes: *Token*, *SimpleClass*, *MetaClass*, and *MetametaClass* (Jarke, 1991, p. 6).

This thesis project investigates the means by which the *GraphBrowser* utility can be extended with the Andrew Toolkit to permit the direct editing of the REMAP model's objects using a graphical user interface.

V. EXTENDING THE GRAPHBROWSER INSET

A. GRAPHBROWSER INSET COMPONENTS

The *GraphBrowser* inset consists of the data object *cbGraph* and the view *cbGraphView*. The class *cbGraph* stores information about the knowledge base objects that comprise the semantic network to be graphically displayed, such as the specific ID of the object within the *ConceptBase* server or the graphical type of the object (Eherer, 1991, p. 1). In addition, the *cbGraph* class provides methods for inserting and deleting objects and computing neighbors of an object. The procedures *cbGraph_Insert*, *cbGraph_Delete*, and *cbGraph_Neighbors* are used for these functions.

The class *cbGraphView* provides methods for displaying the *cbGraph*, such as *cbGraphView_Update* and *cbGraphView_GetInterface*. The class also maintains menu lists which determine the appropriate menu cards to be displayed along with the objects in the semantic network.

In order to provide the capability to graphically access the *ConceptBase* server knowledge base according to the class definitions of the REMAP model, it is necessary to extend the basic *GraphBrowser* functionality.

The following steps are necessary to accomplish this extension:

1. Write the new Andrew class files.
2. Modify the `.graphbrowserinit` file.
3. Modify the knowledge base.
4. Compile the new Andrew class.

These steps are detailed in the following sections.

B. WRITING NEW ANDREW CLASS FILES

1. Class Header File

The class header file, or `".ch"` file, is the class specification and is roughly analogous to standard C include (`".h"`) files. The class header file enables inheritance of procedures from a superclass (ancestor) by defining the class as a subclass, if desired. In the following example, however, no procedures need to be inherited, so the new class will be defined as the top-level class `"issue."`

Example class header file: `issue.ch`

```
#define issue_VERSION 1

class issue {
    classprocedures:
        InitializeClass() returns boolean;
        InitializeObject(struct issue *self) returns boolean;
        FinalizeObject();
};
```

This `.ch` file contains the procedure specification for the initialization of the `issue` class and object instance as well as for cleaning up and freeing memory when the object is deleted. In C++ terminology, this is equivalent to providing specification for class constructors and destructors.

2. Class C File

The corresponding C file contains the actual procedures for which the class was created. The example below illustrates the components of the file `issue.c`.

Example C file: `issue.c`

```
#include "issue.eh"
#include "cbgraphv.ih"
#include "cbGraph.ih"
#include "proctbl.ih"

static void
issue_query(struct cbGraphView *self, long rock)
{
    /* Actual C code of function */
}

boolean issue_InitializeObject(ClassID, self)
struct classheader *classID;
struct issue *self;
{
    return TRUE;
}

void issue_FinalizeObject(ClassID, self)
struct classheader *ClassID;
struct issue *self;
{
}

boolean issue_InitializeClass(ClassID)
struct classheader *ClassID;
{
    struct classinfo *cbGvtype = class_Load("cbGraphView");
    proctable_DefineProc("issue-query", issue_query, cbGvtype,
                        NULL, "Displays all pertinent issues.");
    return TRUE;
}
```

The first section contains the necessary `#include` files, the first of which is `"issue.eh"`. This is the export header file that enables other Andrew classes to utilize the procedures of the `issue` class.

The next section will contain the C code of the function to be performed when the class is invoked through selection of its corresponding menu item from the *GraphBrowser* window.

The *InitializeObject* and *FinalizeObject* are class methods required for proper object instantiation and termination.

Finally, the *InitializeClass* method requires the call to *proctable_DefineProc* which enters the internal procedure name, "issue-query," into the *cbGraphView*'s procedure table. The procedure table, or *proctable*, is used to establish and translate bindings between menu items and their corresponding procedures. The *proctable_DefineProc* method takes the following five parameters, separated by commas (Borenstein, 1990, p. 98):

1. Internal procedure pointer name ("issue-query")
2. Formal C procedure pointer (*issue_query*)
3. Class identifier used in procedure type-checking (*cbGVtype*)
4. Name of module to load procedure from (*NULL*, since procedure is local)
5. Interactive help text (optional)

C. MODIFICATION OF *.graphbrowserinit* FILE

The *.graphbrowserinit* file has two parts: the menu item description part and the graphical type description part. The basic *.graphbrowserinit* file is shown below:

```

#include <gb_init.h>
Show Subclasses~66, GB_NODE, gb-gb_show_subclasses
Show Superclasses~65, GB_NODE, gb-gb_show_supclasses
Show Instances~64, GB_NODE, gb-gb_show_instances
Show classes~63, GB_NODE, gb-gb_show_classes
Show Attributes~62, GB_SOMETHING, gb-gb_show_attributes
Any~61, GB_SOMETHING, gb-gb_any
Erase Link~60, GB_LINK, gb-gb_Erase
Erase Node~60, GB_NODE, gb-gb_Erase

MetametaClass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
MetaClass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
SimpleClass, gray, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
Token, none, oval, GB_NODE|GB_SOMETHING|GB_TOOLS

```

1. Menu Item Description

Each line in the first part is a description of a menu item with the format: menu card, menu item, menu item mask, and menu procedure (Eherer, 1991, p. 8). The menu card is optional and has been omitted from the standard *GraphBrowser* menu item descriptions. The second entry in the menu item description (actually the first item shown in the above example, since the optional menu card identifier was omitted) is the literal string label of the menu item itself. This is the label of the push bar on the menu card that will be clicked on. The number following the tilde (~) determines the relative position of the item on the menu card, with the lower valued-item appearing above the higher-valued items. The menu card from the standard *.graphbrowserinit* file that will appear along with a displayed node will be arranged as follows:

Erase Node
Any
Show Attributes
Show Classes
Show Instances
Show Superclasses
Show Subclasses

The third element in the menu description is the menu item mask, which may be given either as a number directly in the menu item description, or indirectly as an identifier from the file *gb_init.h*, shown below:

```
#define GB_NOTHING      0L
#define GB_NODE         1L
#define GB_LINK         2L
#define GB_SOMETHING    4L
#define GB_UNDO         8L
#define GB_TOOLS        16L
```

To determine exactly which menu items are available with a displayed object, a simple boolean evaluation is performed on the comparison of the menu item mask and the object's mask. The menu item will be displayed only if the result of the following is TRUE:

```
(object mask & menu item mask) == menu item mask
```

As described in the next section, the object mask is defined in the second part of the *.graphbrowserinit* file to allow access to certain menu items when a particular object is displayed.

Note that the result of the bit-wise AND operation above will equal the menu item mask only if the bits set in

the menu item mask are also set in the object mask. An item with mask zero, for example, will appear with all objects, since zero ANDed with any object mask will always yield itself.

The final entry in the menu item description is the internal name of the procedure to be executed when the menu item is selected from the display. The procedure is specified in the same way as the first argument in the `proctable_DefineProc` call, not as specified in the C code of the procedure itself. Notice, therefore, that the name of the class will always be followed by a dash (-).

A menu item labeled "Get Issue," for example, which will call the C procedure `issue_query` from the new Andrew class can be included in the `.graphbrowserinit` file as follows:

```
#include <gb_init.h>
Show Subclasses~66, GB_NODE, gb-gb_show_subclasses
Show Superclasses~65, GB_NODE, gb-gb_show_supclasses
Show Instances~64, GB_NODE, gb-gb_show_instances
Show classes~63, GB_NODE, gb-gb_show_classes
Show Attributes~62, GB_SOMETHING, gb-gb_show_attributes
Any~61, GB_SOMETHING, gb-gb_any
Erase Link~60, GB_LINK, gb-gb_Erase
Erase Node~60, GB_NODE, gb-gb_Erase

/*          APPENDED MENU ITEM AND CARD          */
New Functions~10, Get Issues~1, GB_NODE, issue-query

Metametaclass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
MetaClass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
SimpleClass, gray, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
Token, none, oval, GB_NODE|GB_SOMETHING|GB_TOOLS
```

The item mask of `GB_NODE` (value of 1) will cause this

new item to appear whenever a view mask has the one's bit set. The menu card entry, `New Functions~10`, will generate a menu card labeled "New Functions" positioned underneath the untitled *GraphBrowser* standard menu card. On the New Functions menu card will be a sole menu item, labeled "Get Issues."

2. Graphical Type Description

The second section of the `.graphbrowserinit` file contains the specification of how various classes of objects are to be displayed and what menu items will be available with them. Like the menu item description, the graphical type description consists of four parts separated by commas: class name, fill color of displayed object, shape of displayed object, and object mask. The object mask determines which menu cards and items will be available when the object is displayed. These masks can be logically ORed together to provide flexibility in the definition of various masks.

For example, the inclusion of graphical type descriptions for the objects REQUIREMENT, ISSUE, POSITION, and ASSUMPTION from the REMAP model in the .graphbrowserinit file is illustrated below:

```
#include <gb_init.h>
Show Subclasses~66, GB_NODE, gb-gb_show_subclasses
Show Superclasses~65, GB_NODE, gb-gb_show_supclasses
Show Instances~64, GB_NODE, gb-gb_show_instances
Show classes~63, GB_NODE, gb-gb_show_classes
Show Attributes~62, GB_SOMETHING, gb-gb_show_attributes
Any~61, GB_SOMETHING, gb-gb_any
Erase Link~60, GB_LINK, gb-gb_Erase
Erase Node~60, GB_NODE, gb-gb_Erase
New Functions~10, Get Issues~1, GB_NODE, issue-query

/*          APPENDED REMAP GRAPHICAL TYPES          */
REQUIREMENT, gray, circle, GB_NODE|GB_SOMETHING|GB_TOOLS
ISSUE, none, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
POSITION, none, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
ASSUMPTION, none, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS

Metametaaclass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
MetaClass, black, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
SimpleClass, gray, rectangle, GB_NODE|GB_SOMETHING|GB_TOOLS
Token, none, oval, GB_NODE|GB_SOMETHING|GB_TOOLS
```

To display an object's menu(s) in the GraphBrowser window, the object must first be designated by clicking on it with the left mouse button. Then, when the center mouse button is held down, all menu items will be displayed whose mask bits are also set in the object's mask. When the displayed menu items reside on different menu cards, the multiple cards will be displayed in a staggered offset such that the titles (if any) of the lower cards will be visible. Although only the top card will initially be completely visible, all cards below it may be accessed by pointing to the card's title area.

D. MODIFICATION OF REMAP MODEL

The definition of new graphical types in the .graphbrowserinit file has to be made available to the ConceptBase server by including them in the server model. This is done by Telos modifications to the *X11GraphBrowserNODE* and *X11GraphBrowserEDGE* classes in the file *X11GraphBrowserEDGE.sml*.

First, four additional graphical types must be added as attributes to both the *X11GraphBrowserEDGE* and *X11GraphBrowserNODE* classes. The following example shows the *X11GraphBrowserEDGE* class specifications both before and after the addition of the REMAP graphical type pointers:

Standard GraphBrowser *X11GraphBrowserNODE* class
specification:

Class *X11GraphBrowserNODE* in AnswerRepresentation isA NODE
with

```
graphicalType
  gT1 : MetametaClass;
  gT2 : MetaClass;
  gT3 : SimpleClass;
  gT4 : Token
```

GraphBrowser *X11GraphBrowserNODE* class specification after
the addition of four REMAP classes:

Class *X11GraphBrowserNODE* in AnswerRepresentation isA NODE
with

```
graphcialType
  gT1 : REQUIREMENT;
  gT2 : ISSUE;
  gT3 : POSITION;
  gT4 : ASSUMPTION;
  gT5 : MetametaClass;
  gT6 : MetaClass;
  gT7 : SimpleClass;
  gT8 : Token
```

Notice that the original graphical type pointers of the standard *GraphBrowser* class (*gT1-gT4*) were displaced to the *gT5-gT8* position to correlate with the listed order of graphical types in the modified *.graphbrowserinit* file (shown earlier). If, instead, the REMAP graphical types were added to the end of the *.graphbrowserinit* file as items 5-8, they would correctly be defined by attributes *gT5-gT8* in the *X11GraphBrowserEDGE* and *X11GraphBrowserNODE* class specification.

The second step in modifying the knowledge base model involves ordering the attributes that have been added to the class specification. Since the attributes *gT1-gT8* directly correlate with the listed order of graphical types in the *.graphbrowserinit* file, it is necessary only to add the *orderValues* 5-8 to *gT5-gT8*, respectively.

The addition of `orderValue` definitions to the `X11GraphBrowserNODE` class is shown below:

```
X11GraphBrowserEDGE!gT1 with  
orderValue  
    v:1  
end
```

```
X11GraphBrowserEDGE!gT2 with  
orderValue  
    v:2  
end
```

```
X11GraphBrowserEDGE!gT3 with  
orderValue  
    v:3  
end
```

```
X11GraphBrowserEDGE!gT4 with  
orderValue  
    v:4  
end
```

```
X11GraphBrowserEDGE!gT5 with  
orderValue  
    v:5  
end
```

```
X11GraphBrowserEDGE!gT6 with  
orderValue  
    v:6  
end
```

```
X11GraphBrowserEDGE!gT7 with  
orderValue  
    v:7  
end
```

```
X11GraphBrowserEDGE!gT8 with  
orderValue  
    v:8  
end
```

Standard `orderValue`
representation of the
`X11GraphBrowserEDGE`
attributes `gT1-gT4`

Appended `orderValue`
representation of
attributes `gT5-gT8`

Note that the same modifications must be performed to the *X11GraphBrowserEDGE* and *X11GraphBrowserNODE* specifications. If the *gT* attribute order does not correlate with the order in the *.graphbrowserinit* file, the *orderValue* definition can be used to provide an accurate cross-reference between the two lists. In order to avoid confusion, however, it is best to arrange the graphical type definitions in the *.graphbrowserinit* file in the same order as the *gT* attributes of the *X11GraphBrowserNODE* and *X11GraphBrowserEDGE* class specifications.

E. COMPILATION OF THE ANDREW CLASS

The C-code compilation associated with this thesis research was conducted using *Imakefile* macros developed specifically for Andrew systems development efforts. An *Imakefile* is a *Makefile*-generator that consists of a set of templates containing rules that are expanded into a *Makefile* customized for specific system types and configurations. The *imake* program, created by Todd Brunhoff of Project Athena and Jim Fulton of the X Consortium, passes the *Imakefile* through the C preprocessor, *cpp*, to produce a *Makefile* with applicable file descriptions and dependencies (Oram, 1991, p. 111).

The overall file compilation effort can be summarized in the following steps, explained in the subsequent paragraphs:

1. Set environment variables.
2. Create *Imakefile*.
3. Installation of *Imakefile*, *.c* and *.ch* file in project directory.
4. Generate *Makefile* and compile C code.

1. Setting Environment Variables

It is first necessary to ensure that the environment variables in the *.cshrc* file of the project account are properly set to look for executable and dynamic objects within the account directory. The following environment variables should be set to build and test Andrew code:

```
setenv ANDREWDIR /usr/local/andrew
setenv PATH .:$ANDREWDIR/bin:$PATH
setenv CLASSPATH .:$ANDREWDIR/dlib/atk
```

2. Creating the *Imakefile*

The following *Imakefile* was created using the emacs editor and installed in the project directory:

```
NormalObjectRule()
DependTarget()
NormalATKRule()
DynamicObject(remap,,)
InstallClassFiles(remap.do, remap.ih)
CC = gcc
PICFLAG = -fpic
```

The *NormalObjectRule* is a standard default dependency rule used whenever *.c* files need to be compiled into *.o* files and eventually into *.c* files. *DependTarget* sets up directory dependencies, while the *NormalATKRule* establishes dependencies for the production of *.ih*, *.eh*, and *.do* files. The *DynamicObject* macro is used to create *.do* files which depend upon only one object file of the same basic name. The file

name of the dynamic object to be created is listed as the only argument (without the *.do* extension). Optionally, a space-separated list of library files to be linked against can be included as the second argument. The final macro shown is the *InstallClassFiles*, used for installing dynamic object (*.do*) and associated class header (*.ch*) and import header (*.ih*) files. Required arguments are the *.do* files and corresponding *.ih* files to be installed.

Since the standard C compiler, *cc*, does not accept the function prototypes in the *GraphBrowser* include files, it's best to use the ANSI compiler, *gcc*, by including the rule *CC = gcc* in the *Imakefile*. This will eliminate the appearance of many confusing errors at compilation.

The last line in the *Imakefile*, "*PICFLAG = -fpic*," is necessary since the *gcc* compiler uses the option *-fpic* to get position-independent code and does not recognize the *-pic* option normally generated by *imake*.

3. Installation of Project Files

The next step in preparation for the Andrew class compilation is to ensure the appropriate files are installed in the project directory. For simplicity's sake, the project account's home directory was used as the project directory for this thesis, but a dedicated subdirectory could have been created to segregate the project code from unrelated files in the home directory. The necessary files to be installed are

the *Imakefile*, the modified *.graphbrowserinit* file, the *.c* file, and the class header (*.ch*) file.

4. Makefile Generation and Code Compilation

The final step in the compilation process is to generate the *Makefile* and compile the code for dynamic loading by the *GraphBrowser*. With the project directory containing the above mentioned files set as the current working directory (*cwd*), type the command

```
% genmake
```

to generate the *Makefile* which will appear in the *cwd*. When the prompt returns, type

```
% make depend
```

to set up the file dependencies. Finally, type

```
% make
```

to generate and index the binary *.do* file which will be also loaded in the *cwd*. The *.c* file can be re-compiled after modifications to the source code by omitting the first two commands and simply typing

```
% make
```

The functions contained within the newly compiled ATK class can be accessed by clicking on their corresponding menu items in the *GraphBrowser* window. The *GraphBrowser* can be initiated from the *ConceptBase Toolbar* as described in the Appendix.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. HIGH-LEVEL VERSUS LOW-LEVEL TOOLKITS

A drawback to using high-level toolkits is that an application ported across different window managers will have an inconsistent look and feel. Applications written completely in the low-level Xlib interface have the advantage of being consistently portable across all standard X Window implementations, but they can be tedious and difficult to implement. Hundreds of lines of code can be needed to handle the window manager conventions alone (Young, 1990, p. 11). X toolkits provide a set of utility functions and prefabricated user-interface components that allow the developer to focus on the interface design. Although most X toolkits are based on Xlib routines, they are generally much easier to use than Xlib, since implementation details are hidden from the programmer.

B. SELECTION OF THE ANDREW TOOLKIT

The multi-media potential of Andrew Toolkit applications made Andrew an attractive choice for the conversion of the Sun View REMAP application to an X Window interface. The capture of process knowledge by the REMAP model will be facilitated by Andrew's multi-media capacity, since software design engineers' deliberations will be manifested in many forms -

flow charts, structure charts, data flow diagrams, bit-mapped and raster images, text, and imbedded audio and video. This rationale for using Andrew is reinforced by the fact that ConceptBase, the REMAP prototype environment, is itself written in Andrew. ATK was chosen for ConceptBase since, like REMAP, it was intended as a coordination mechanism for heterogeneous design environments.

C. LEARNING ANDREW

1. Prerequisite Skills

A set of prerequisite skills must be acquired before initiating an Andrew Toolkit development project. First, a fluency with the development platform's operating system and window manager are essential. A working familiarity with the C language is also necessary, with special emphasis on structures, pointers, and arrays. Next, the basics of the X Windows environment must be understood, including the client-server model and the function of the X window manager. With this background, the role of the high-level toolkit and its relationship with the Xlib C language interface will become clear. Finally, "hands-on" experience with the application under development is required for a thorough understanding of the required functionality. In the event that the development is a conversion project, the current implementation must also be well understood, since it will serve as a template for the design of the target application.

2. Andrew-specific Skills

Apart from Nathaniel S. Borenstein's well-written reference, *Multimedia Applications Development with the Andrew Toolkit*, virtually all reference material on Andrew Toolkit programming is in the form of on-line documentation within the X11 distribution. On-line sample code from the examples used by Borenstein combine with the book to provide a valuable tutorial. The Andrew source code can also be a helpful reference and can be FTPed from *emsworth.andrew.cmu.edu*, *userid: anonymous*, under the *split/* directory in files *andrew.aa.tar.z* through *andrew.aj.tar.z*. The Remote Andrew Demo Service, provided by the Andrew Toolkit Consortium at Carnegie Mellon University, is a good way to experience Andrew applications firsthand. To use the remote demo, log on to a workstation running X11 with access to the Internet.

Run the commands:

```
finger help@atk.itc.cmu.edu
xhost +atk.itc.cmu.edu
finger run-demo@atk.itc.cmu.edu
```

Further information on Andrew may be available from the Andrew Toolkit Consortium at Carnegie Mellon University:

Andrew Toolkit Consortium
Carnegie Mellon University
4910 Forbes Avenue
Pittsburgh, PA 15213-3890
(412) 268-6700 / FAX: (412) 621-8081

Mailing list: info-andrew@andrew.cmu.edu
Newsgroup: comp.soft-sys.andrew
ATK Consortium info: wjh+@andrew.cmu.edu

D. CLOSING REMARKS

Although the current X11 GraphBrowser application does not have the full functionality of the Sun View REMAP implementation, the necessary extension to the X11 GraphBrowser can be achieved with relatively little effort using the Andrew Toolkit.

APPENDIX

RUNNING THE GRAPHBROWSER IN THE X11 USAGE ENVIRONMENT

Running the GraphBrowser from a Remote Workstation

1. Login and run X Windows by entering `<startx>` or `<xinit>`.
2. In console window, enter `<xhost +>`. The advisory "all hosts being allowed (access control disabled)" will appear.
3. In an X terminal, rlogin to the workstation where the *GraphBrowser* and extension is installed. This terminal will hereafter be referred to as the GB terminal.
4. In the GB terminal, enter `<setenv DISPLAY <local machine address>:0.0>` to send *GraphBrowser* graphics to the local workstation.
5. In another X terminal rlogin to the machine running *ConceptBase*. This terminal will be referred to as the CB terminal.
6. Continue with "Procedures Common to Local and Remote *GraphBrowser* Operation."

Procedures Common to Local and Remote GraphBrowser Operation

1. Start the *GraphBrowser Toolbar* in the GB terminal by entering `<$CB_HOME/X11_UE/'arch'/toolbar>`. The Toolbar window should appear within a few seconds.

2. In the CB terminal, start the *ConceptBase* server by entering `<$CB_HOME/goCBserver>`. When the server is started, note the portnumber that the server is "ready under" (typically 4001).
3. In the GB terminal, display the SYSTEM menu by first clicking on the SYSTEM box with the left mouse button to designate the SYSTEM tool, then point and hold with the center mouse button to pull down the menu card.
4. From the SYSTEM menu card select "connect CB server" by releasing the center mouse button on the "connect CB Server" menu item while backlit.
5. Designate the MODELS tool by clicking on its box with the left mouse button, and pull down the menu card by holding down the center mouse button. Select "load application" by releasing center mouse button on backlit item.
6. An *Interaction Window* will appear and ask the pathname. Enter `</files/is1/cbase/project/>`. For application name, enter `<MyApp>`. When the word "Done" appears in the *Toolbar* comment area, the application has been loaded.
7. From the BROWSING menu card, select *GraphBrowser* using the technique described in steps 4 and 5 above. An *Interaction Window* will appear and request the name of the object to browse. Enter `<process_data>`. The comment "The GraphBrowser starts up. Wait a little bit for its window." will appear in the *Toolbar* comment area.

8. The *GraphBrowser* window will appear with the `process_data` object graphically displayed. Operations on displayed objects can be performed by selecting the object with the left mouse button, then holding down the center button to display the applicable menu cards from which the desired function can be selected.

LIST OF REFERENCES

- Barkakati, Nabajyoti, *X Window System Programming*, Sams, 1991.
- Borenstein, Nathaniel S., *Multimedia Applications Development with the Andrew Toolkit*, Prentice Hall, Inc., 1990.
- Eherer, Stefan, and Baumeister, Markus, "Documentation of the Andrew class cbGraph and cbGraphView and how to extend the functionality of the standard GraphBrowser," addendum to *ConceptBase V3.0 User Manual*, University of Passau, 1991.
- Jarke, Matthias, *ConceptBase V3.0 User Manual*, University of Passau, 1991.
- Jones, Oliver, *Introduction to the X Window System*, Prentice Hall, 1989.
- Oram, Andrew, and Talbott, Steve, *Managing Projects with Make*, O'Reilly & Associates, 1991.
- Palay, Andrew J. and others, "The Andrew Toolkit: An Overview," paper presented at the USENIX Association Winter Conference in Dallas, Texas, February 1988.
- Ramesh, Balasubramaniam, and Dhar, Vasant, "Supporting Systems Development Using Knowledge Captured During Requirements Engineering," *IEEE Transactions on Software Engineering*, June 1992.
- Upton, Molly, "Is X the Window?," *Patricia Seybold's Office Computing Report*, pp. 1-8, May 1990.
- Young, Douglas A. *The X Window System, Programming and Applications with Xt*, Prentice Hall, 1990.

BIBLIOGRAPHY

Borland International, *Turbo C++, Getting Started*, 1990.

• Gilly, Daniel, and O'Reilly, Tim, *The X Window System in a Nutshell*, O'Reilly & Associates, 1990.

• Kernighan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, 1978.

Massachusetts Institute of Technology, X11R5 online documentation, 1991.

Meier, Carol, "An Introduction to C," tutorial notes, January 20, 1992.

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 52
Naval Postgraduate School
Monterey, California 93943-5002 | 2 |
| 3. | B. Ramesh, Code AS/RA
Naval Postgraduate School
Monterey, California 93943 | 3 |
| 4. | Roger Stemp, Code CS/SP
Naval Postgraduate School
Monterey, California 93943 | 1 |
| 5. | LCDR Jeffrey J. Stenzoski
3963 Devonshire Drive
Marietta, Georgia 30066 | 2 |